1-1-2002

Combined scheduling of hard and soft real-time tasks in multiprocessor systems

Basheer Nayef Al-Duwairi
Iowa State University

Combined scheduling of hard and soft real-time tasks in multiprocessor systems

by

Basheer N. Al-Duwairi

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Manimaran Govindarasu, Major Professor
Daniel Berleant
Simanta Mitra

Iowa State University

Ames, Iowa

2002

Graduate College
Iowa State University

This is to certify that the master's thesis of

Basheer N. Al-Duwairi

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOLEDGMENTS

I would like to express my sincere thankfulness to my parents, brothers and sisters in Jordan for their continuous encouragement and support.

I am deeply indebted to my supervisor Dr. G. Manimaran whose help, stimulating suggestions and encouragement helped me in all the time of research and for writing of this thesis. Special thanks to Dr. Daniel Berleant and Dr. Simanta Mitra for dedicating their time to participate in my graduate committee.

I would also like to thank S. Swaminathan for helping me on several occasions with valuable advice and suggestions.

To all my friends at Iowa State University for making me feel at home, and for their help and advice.

ABSTRACT

Many complex real-time systems are composed of both hard and soft real-time tasks. Combined scheduling of hard and soft tasks in such systems should satisfy two important goals: (1) maximize the schedulability of soft real-time tasks with no or little impact on the schedulability of hard real-time tasks; (2) minimize the scheduling overhead. In this thesis, we develop two sets of algorithms for the problem, of which the first set allows sacrificing the schedulability of hard tasks and the second set does not. The first set of algorithms is based on a new concept, called "task association", by which each soft task is associated with a hard task, whenever possible, in order to minimize the scheduling overhead. The second set has two algorithms, namely, background scheduling and emergency based scheduling. The background scheduling schedules soft tasks in the holes that are present in the schedule considering only the hard tasks. The emergency based scheduling always maintains two schedules (primary schedule and emergency schedule) and switches back and forth between them during the schedule construction process depending on the schedulability of a given hard task. To evaluate the schedulability of the proposed algorithms, extensive simulation studies were conducted and the results show that the proposed algorithms are superior to existing algorithms, in addition to some of them incurring lesser scheduling overhead.

CHAPTER 1: INTRODUCTION

1.1 Real-Time Systems

Real-time computing is a key enabling technology for the future in an-ever growing domain of important applications [21]. This area covers a wide spectrum of applications and systems. This includes nuclear plants, robot control systems, air defense systems, multimedia applications, image and speech processing, autonomous vehicle control, and computer driven automatic control systems. Real-time systems are those systems in which the correctness of the output depends not only on the logical result of the computation, but also on the time by which the results are produced [13]. In real-time systems tasks are assigned deadlines. Failing to meet the deadline can result in catastrophic or undesirable consequences depending on the nature and role of the task in the system.

As shown in Figure 1, real-time systems consist of controlling system and controlled system both interacting in the same environment. The controlled system is monitored continuously by sensors. Timely actions are activated by the controlling system based on the information it receives from the sensor. For example in an air defense system, when a threat is detected and confirmed, the engagement task is activated, resulting in the firing of a missile to engage the threat. After the missile is in flight, the missile guidance task uses sensor data to track the threat, and issues guidance commands to the missile.

Real-time systems need to be predictive, reliable, and fast. Predictability is the ability to determine for a given set of tasks whether the system will be able to meet all of the timing requirements of those tasks. Predictability calls for the development of scheduling models and analytical techniques to determine whether or not a real-time system can meet its timing requirements. On the other hand, reliability ensures that all critical tasks always make their deadlines (a 100% guarantee), subject to certain failure and workload assumptions. Such guarantee is usually given by an offline analysis and by schemes that reserve resources for these tasks. Being fast is a necessary condition for real-time systems, however it is not

sufficient. Recall that the main objective of a real-time system is to meet explicit deadlines and being fast on the average does not guarantee that a deadline will be met.

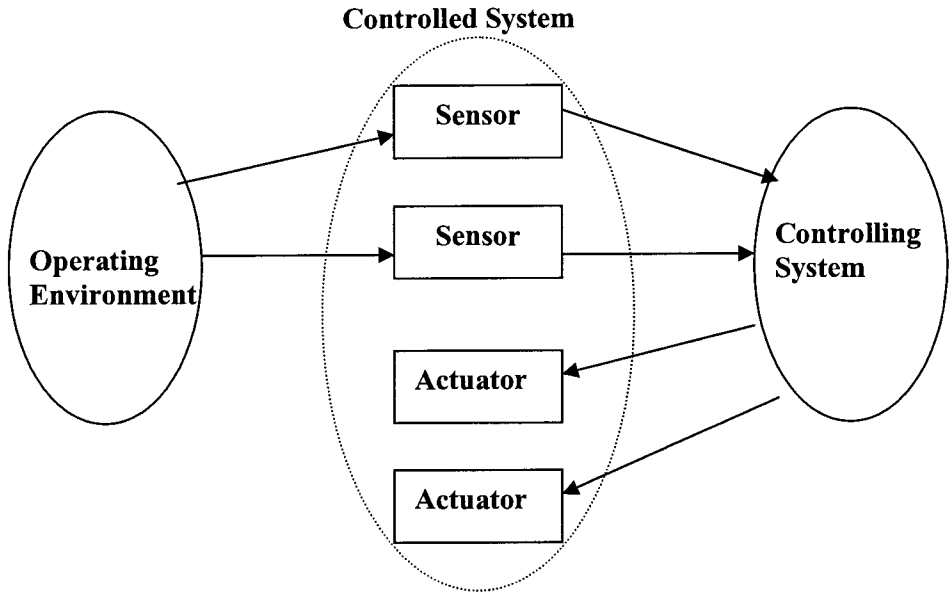


Figure 1. A Typical Real-Time System

1.2 Tasks in Real-Time Systems

Tasks can be classified based on the nature of their activation as *periodic* or *aperiodic*, or based on their criticality as *hard*, *firm*, or *soft*. In general, any task is characterized by many parameters that must be considered when allocating time and space (i.e., processors and resources) for them to execute. These parameters include the worst case computation time, deadline or period, criticality value or importance level that imposes special handling of such tasks, and resource and precedence constraints that can prevent the execution of some tasks even if some resources are available.

1.2.1 Activation Time Classification

Two types of tasks that are commonly encountered in real-time systems designed for monitoring and control functions. These are *periodic* tasks and *aperiodic* tasks [17]. In this

section we highlight the differences between the two types.

Periodic Tasks

Such tasks are activated at regular intervals. Each task is characterized by its computation time C and its period P. The deadline of certain instance arriving at time t is $t + P$. To monitor a physical system or process, a computer system must sample it and react to the data gathered. This regular sampling gives rise to periodic task. For example, processing activities related to sensory acquisition and low-level control must be periodically executed to ensure a correct reconstruction of external signals and guarantee smooth and stable behavior of the controlled system.

Aperiodic Tasks

Such tasks are triggered at irregular intervals when some particular conditions occur. Each task is characterized by set of parameters that are not known a priori, such as its arrival time, worst case execution time, and a deadline. Planning special actions, failure recovery, and handling exceptional situations do not require periodic execution, so they fall in this category.

1.2.2 Criticality Based Classification

Three types of tasks can be identified based on their criticality level. Each task contributes by certain value to the system upon its execution. The relationship between value and time is shown in Figure 2 for the three types.

Hard Tasks

Here the deadline is critical, and it is imperative that it has to be met in all working scenarios to avoid catastrophic consequences. Many safety-critical systems are hard real-time systems. Such systems generally rely on a priori knowledge about the applications' worst

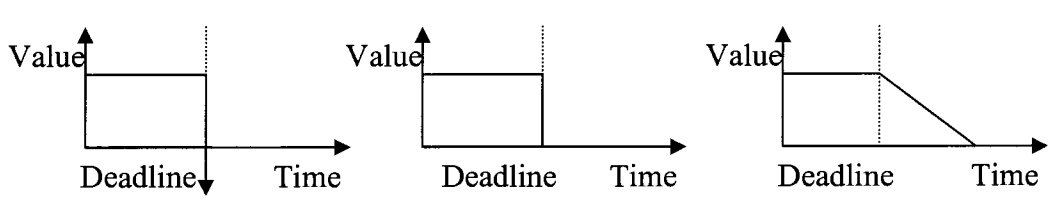
case resource needs to determine a feasible schedule or a set of priorities that will guarantee that the deadlines will be met. Examples of hard real-time systems include embedded tactical systems for military applications, flight mission control, traffic control, robotics, nuclear plants, etc.

Firm Tasks

The result produced by such tasks is not useful as soon as the deadline expires. Missing the deadline can be tolerated. However, it is not desirable because such deadline misses lead to wasting of system resources without contributing any value to the system. Online transaction processing applications, such as airline reservation and banking are examples of firm real-time systems.

Soft Tasks

These tasks need only a reasonable assurance that their resource needs will be met by the system. Failure to meet a deadline does not lead to system or application failure, it is simply less satisfactory to the user. For example, desktop video playback is a non-critical real-time application. It has deadlines associated with the playback of individual frames of video, but failure to meet those deadlines results in lower user satisfaction rather than outright failure of the application.



(a) Hard deadline (b) Firm deadline (c) Soft deadline

Figure 2. Different Types of Deadlines

1.3 Scheduling in Real-Time Systems

Scheduling involves the allocation of resources and time to tasks in such a way that certain performance requirements are met. This is a multidimensional problem in which many parameters and constraints must be considered. The central problem in multiprocessor scheduling is to determine when and on which processor a given task executes. Scheduling in general can be static or dynamic. Static scheduling refers to the fact that the scheduling algorithm has complete knowledge regarding the task set and its constraints such as deadlines, computation times, precedence constraints and future release times [22]. Static scheduling algorithm operates on such task set and produces a single schedule that is fixed for all the time. In contrast, a dynamic scheduling algorithm has a complete knowledge of the currently active task set, but new arrivals may occur in the future, not known to the algorithm at the time it is scheduling the current task set [24]. Therefore the schedule keeps changing over time. In fact, many scheduling problems are NP-complete [22]. As a result, heuristics algorithms are used for such scheduling problems. A real-time scheduling policy has two components 1) the schedulability test, which is used in the design phase, and 2) the run-time scheduling algorithm itself. The goal of the first component is to answer the question of whether the scheduling algorithm properly schedules the tasks of the system. The second component is the algorithm that arranges the task execution while the system is running.

Considerable research efforts enriched the field of real-time scheduling [7], [8], [10], [11], [12], [13], [14], [20] and [21]. Under static and dynamic scheduling, Ramamritham and Stankovic presented four scheduling paradigms in [13]. These paradigms are the *static table-driven approaches*, *priority driven preemptive approaches*, *dynamic planning based approaches*, and *best effort approaches*. This classification depends on (a) whether a system performs schedulability analysis, (b) If it does, whether it is done statically or dynamically, and (c) whether the result of the analysis itself produces a schedule or plan according to which tasks are dispatched at run-time. In the following subsections we discuss the above scheduling paradigms in more detail, and provide examples about each paradigm.

1.3.1 Static Scheduling

Static Table Driven Approaches

Both schedulability check and schedule construction are done offline. This approach is applicable to periodic tasks with known characteristics, the start time and completion time of each task are determined offline for the LCM (least common multiple) using heuristic search algorithms if the tasks have resource or precedence constraints and the system has multiple processors. The resulting schedule is stored in a table to be used at run time. This is a highly predictable approach that pre-allocates all resources needed to meet the deadlines of safety-critical tasks a priori. However, it has a major problem; any change in the task set, including modification of the tasks themselves, can require reconstructing and fully testing a new schedule.

Priority-Driven Preemptive Scheduling

The schedulability check is done offline, but the schedule is constructed online. Under this approach, tasks are dispatched according to their assigned priorities. At any given time, the task with highest priority executes, if a higher priority task enters the system then it will preempt the currently executing task and start execution. Priority assignment is done either statically or dynamically. Static priority assignment is used for periodic tasks. It is simple to implement because a task's priority is assigned once it arrives and does not have to be reevaluated as time progresses. Dynamic priority assignment on the other hand can be used for both periodic and aperiodic tasks. It is more expensive to use in terms of run-time overheads, because a task's priority may change if other tasks with higher priority arrive.

Static Priority-Driven Preemptive Scheduling

Rate Monotonic Scheduling (RMS)

Traditionally, the requirements of a periodic real-time task are characterized by a period P, and a worst-case computation time C. The utilization of a task is defined to be (C / P). The

problem of scheduling such tasks was first addressed by Liu and Layland [8], when they introduced an optimal fixed priority-scheduling algorithm called the rate monotonic scheduling (RMS). Which later became one of the most widely used scheduling policies for preemptive periodic real-time tasks on uniprocessor systems. RMS associates each task with priority equals to ($1/P$). At any time the available instance with the highest priority is being processed, where an available instance is one that has been released and has not yet completed. It is assumed that a task can be preempted by a higher priority task in negligible time. This scheme is optimal in the sense that if any static priority scheme can schedule a given set of periodic tasks then the rate monotonic algorithm can [24]. The schedulability test used in RMS is given by

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{1/n}-1)$$

This test shows that a task set of size n will be able to meet all the deadlines all the time if the rate monotonic algorithm is used and the total utilization is not greater than $n(2^{1/n}-1)$, a quantity which decreases monotonically from 0.83 when $n = 2$ to 0.693 as $n \rightarrow \infty$. However, this condition is sufficient but not necessary, which means that a task set with total utilization greater than the given bound and lower than 1 can still be schedulable under RMS. This motivated the work towards developing exact analysis for the RMS. Exact analysis of the schedulability under RMS is presented in [7].

Dynamic Priority-Driven Preemptive Scheduling

Earliest Deadline First (EDF)

EDF is an optimal scheduling algorithm under dynamic priority assignment for uniprocessor systems proposed by Liu and Layland [8]. They focused on scheduling set of independent periodic tasks, each is characterized by computation time C and period P, showing that full processor utilization is always achievable, and giving a simple test that is both necessary and sufficient for scheduling a set of tasks. The test is called the utilization test and is given by

$$\sum_{i=1}^{n} c_i / p_i \leq 1$$

This algorithm gives the highest priority to the task with the least deadline.

Least Laxity First (LLF)

LLF is another optimal dynamic priority scheduling algorithm for uniprocessor systems. The highest priority is given to the task with the least laxity (laxity is defined as the amount of time task can wait and still meet its deadline).

1.3.2 Dynamic Scheduling

Dynamic Planning Based Scheduling

In this approach task feasibility and schedule construction are determined at run time. A plan for task execution is constructed based on different task parameters like arrival time, worst-case execution time, deadline, and resource usage. The key issue here is the ability to give deterministic guarantee for a task that it will meet its deadline once accepted by the scheduler. This approach was adopted in Spring scheduling (also known as Myopic algorithm) [12] described below, and its variants [10] and [11].

Myopic Algorithm

Myopic is a heuristic search algorithm for scheduling tasks with resource constraints in multiprocessor systems. This algorithm uses a tree structure in which a vertex is a partial schedule, and a leaf is a complete schedule. Given a set of tasks with resource requirements, myopic algorithm tries to determine a full feasible schedule by starting at the root of the search tree which is an empty schedule and tries to extend the schedule by moving to one of the vertices at the next level in the search tree until a full feasible schedule is derived. The algorithm evaluates a heuristic function for k tasks at a time, and extends the schedule by the task with minimum heuristic value. If a partial schedule is found to be infeasible, it

backtracks and then continues the search. This algorithm was implemented in the Spring kernel [20].

Dynamic Best effort scheduling

In this approach, all newly arriving tasks are admitted by the system without schedulability check. The scheduler then tries its best to guarantee the execution of these new tasks. It is used to schedule tasks at times of overload, where some tasks have to be rejected. A good algorithm rejects as minimum tasks as possible. In this paradigm, tasks are assigned priorities based on RMS, EDF, or LLF polices. Also each task may have importance level, or criticality value. When the system is under loaded, the best effort scheduler schedules tasks based on their assigned priorities where the arrival of high-priority task preempts a currently executing low-priority task. On the other hand, if the system is overloaded, the objective becomes to maximize the overall value contributed to the system by the chosen tasks. Lowest-value-density-first (LVDF) is an example of task discarding policy during overloads.

1.4 Motivation for Combined Scheduling of Hard and Soft Real-Time Tasks

The problem of jointly scheduling hard and soft real-time tasks becomes increasingly important. With phenomenal improvement of hardware technologies in recent years, workstations and PC desktops are becoming increasingly popular platforms for real-time applications such as multimedia audio and video. Many industrial and military applications with real-time computing demands are composed of tasks of various types and constraints. For example automated manufacturing and attack helicopters are being designed to take advantage of audio and video information [4]. It is possible to use completely separate platforms to implement different types of systems, however replacing such redundant systems with an integrated system that can deal with hard and soft real-time tasks, can reduce the cost, since the reduction in the number of cables, display equipments, etc. is significant. In addition, an integrated platform can provide more functionality. An integrated platform which is capable of meeting the requirements of such mixture of tasks is highly desirable.

1.5 Contribution of this Thesis

The work presented in this thesis is a continuation to the research efforts in the area of combined scheduling of hard and soft real-time tasks. One contribution of this thesis is an association based combined scheduling algorithm with the primary objective of reducing the high scheduling overhead while providing a notion of fairness for soft tasks by assigning a fraction of CPU time for them. Other contribution is represented by the proposed *background* and *emergency* based combined scheduling algorithms, which are developed to provide a deterministic guarantee that schedulability of hard tasks will not be affected by the existence of soft tasks in the system, with a secondary objective to schedule as many soft tasks as possible. This thesis gives detailed descriptions about these algorithms, their implementations, and performance studies.

1.6 Organization of the Thesis

Chapter 2 describes several research efforts proposed in the literature to address the problem of combined scheduling for different task models and applications. In chapter 3 the proposed scheduling approach is discussed and details about the task and system model, association policy, association algorithms, and performance studies are presented. In chapter 4, two combined scheduling algorithms (the *background-based* and *emergency-based* combined scheduling algorithms) are proposed. Chapter 5 presents the conclusions drawn from the experimental results, and suggests several future research directions.

CHAPTER 2: RELATED WORK

2.1 Introduction

In combined scheduling of hard and soft real-time tasks, minimizing scheduling overhead becomes primary objective when dealing with large number of tasks of both types, soft and hard. Because while the scheduler is busy in making scheduling decisions for soft tasks, some waiting hard tasks may miss their deadlines. This issue motivated the need for developing efficient schemes that is able to provide real-time support for soft tasks in a hard real-time system, while the total number of tasks considered for scheduling is reduced. This goal was introduced in [4] and [2] where the *multimedia server mechanism* was used to accommodate soft tasks. Another objective in such systems is to provide a deterministic guarantee that no hard task will miss its deadline because of any other soft task in the system, while accepting as many soft tasks as possible. Similar objective with the concentration on minimizing the average response time of soft tasks was achieved by the *aperiodic-online assignment* and *global aperiodic scheduling* schemes presented in [16]. Although a lot of attention has been paid to combined scheduling of hard and soft real-time tasks in uniprocessor systems [1], [3], [5], [6], [15], [18] and [19], very little research has been done in this area for multiprocessor systems [4], [2], and [16]. In this chapter we present the related efforts in multiprocessor systems.

2.2 Multimedia Server

In [4] Kaneko et al. introduced the concept of *multimedia server* to support the co-existence of multimedia applications and traditional hard real time applications that interact via shared use of CPUs. Their work was inspired by the need of flexible and dynamic scheduling that includes various types of interaction between the hard and soft real time control tasks. Hard real-time applications such as automated manufacturing and attack helicopters are examples of such applications where they were designed to take advantage of audio and video information via multimedia streams.

The *multimedia server* is a periodic task that is created dynamically and scheduled along with hard real-time tasks. It is used to accommodate multimedia tasks for the purpose of reducing the total number of tasks considered by the planning based scheduler, because considering each multimedia task instance as hard real-time task and scheduling it without having the multimedia server will increase the scheduling cost. In their work, two allocation mechanisms by which multimedia task instances are assigned to server instances were proposed, the *proportional allocation* and the *individual allocation*.

2.2.1 Proportional Allocation

In this scheme, the multimedia server is a periodic stream with period P_s equal to the smallest period of all multimedia streams in the system. Assuming that there are n different multimedia streams in the system, stream i is characterized by its period P_i and execution time C_i. Then, since each task instance is divided into P_i/P_s server instances, the computation time of the multimedia server instance C_s is given as

$$C_s = \sum_{i=1}^{n} \left(C_i \frac{P_s}{P_i} \right).$$

Clearly, the time complexity for this scheme is $O(n)$, where n is the number of different multimedia streams in the system, because the stream with the smallest period out of n streams is to be found. Figure 3 illustrates this allocation scheme. As multimedia stream 1 has the shortest period P_1, the server has the same period as stream 1, namely P_1. In this example, the length of each server instance is the sum of the execution times of the task of stream 1, half of stream 2 and one third of stream 3.

2.2.2 Individual Allocation

In this approach, as shown in Figure 4, multimedia stream instances are assigned individually to server instances. The period of the server is the same as the minimum period of all multimedia streams multiplexed into the server. Each stream task instance is assigned to the nearest server instance that is located between its release time and deadline. If such

server instance cannot be found, a new server instance has to be created. Again the time complexity for this scheme is *O (n),* since the stream with the smallest period is to be found.

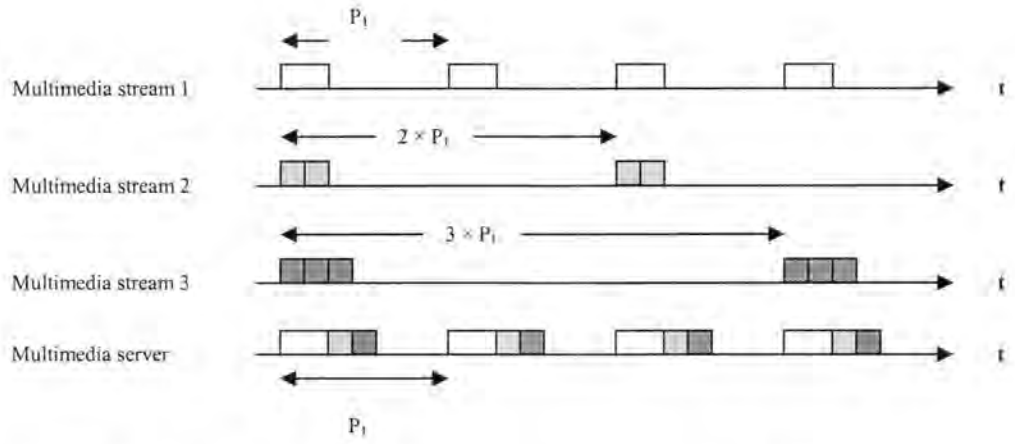


Figure 3. Proportional Allocation of Multimedia Streams

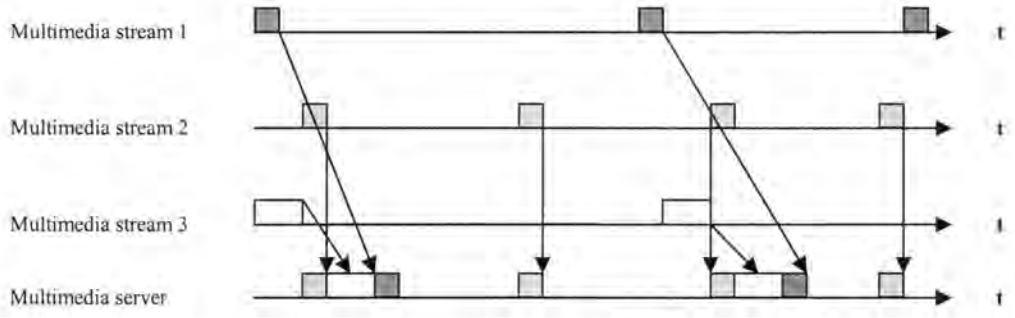


Figure 4. Individual Allocation of Multimedia Streams

2.2.3 Drawbacks of Multimedia Server

By using the *multimedia server* to accommodate periodic soft tasks, the authors in [6] ignored that all individual stream task instances belonging to the same stream cannot be guaranteed to meet their deadlines, even if some deadline misses can be tolerated for each stream, how many and which instances will miss their deadlines is not known under the planning based scheduler. This leads us to believe that controlling the quality of each stream is not possible. The *proportional allocation* scheme presents a clear problem, assigning task

instance's computation time proportionally to many server instances, requires that all server instances to which the task is mapped to meet their deadlines in order for this task to meet its deadline, which is difficult to guarantee. And if one of the server instances misses its deadline, which is more likely to happen then the CPU time allocated for the same task with other server instances is wasted even if the other server instances found to meet their deadlines. In addition to the problems presented above, the *multimedia server* has several other drawbacks. In *proportional allocation*, the context switch overhead can reduce the performance. Under certain circumstances especially when there is large number of multimedia streams, it is impossible for the multimedia server to accommodate all streams because the server size could be larger than its period. In other circumstances the server instances could have very tight laxity, and hence they are expected to be not feasible, which results in poor performance. Also the order by which the streams are executed inside the server is not determined. Although the other approach -*the individual allocation*- presents solutions to some of the problems mentioned above especially the context switch problem, large sized server instances affect the schedulability of hard tasks in the system. This calls for new allocation mechanisms that eliminate the above problems and provide better performance at the same time.

2.3 Integrated Scheduling With QoS Degradation

In [2] Anita et al. developed a scheduling algorithm based on the *multimedia server* concept presented in [4] to schedule both soft periodic tasks and aperiodic hard real-time tasks. In their work, they gave attention to the QoS degradation issue by using the (*m, k*) model [14], and the imprecise computation model [9] to characterize the soft load. Each multimedia stream was characterized by the *m, k* parameters. After mapping each multimedia task instance to a suitable server instance, the resulting server instances were characterized by the imprecise computational model, where the computation time of each instance is divided into two parts: one is mandatory and the other is optional based on the *m, k* parameters of tasks used to construct that particular server instance. All server instances were considered as hard real-time tasks and scheduled with other hard tasks using myopic algorithm. Two types of algorithms were described: *0/1 degradation,* in which a task set is

tested for schedulability with maximum quality by considering all optional parts. If the set is found to be unfeasible, scheduling with minimum quality by ignoring the optional parts is done. The other type of algorithms is the *multilevel degradation* algorithm, in which the quality is degraded in steps by reducing the optional parts considered each time. Preliminary test based on minimum or maximum quality was conducted to see if the set can be scheduled or not before using the actual scheduling algorithm. Although they were able to nicely control the quality of the streams, they ignored the high scheduling cost.

2.4 Approaches to Guarantee Hard Real-Time Tasks

In [16] two approaches were proposed for jointly scheduling of both hard deadline periodic tasks and soft aperiodic tasks in multiprocessor systems. Their objective was to guarantee all hard real-time tasks, with concentration on minimizing the response time of soft aperiodic tasks. The basic assumption was that soft tasks have no deadlines at all. In both approaches periodic tasks are assumed to be independent with no resource constraints, and are pre-allocated to available processors based on certain partitioning algorithm. The following sub-suctions discuss both approaches.

2.4.1 Aperiodic On-line Assignment

In this approach the aperiodic task assignment is performed at runtime. Upon the arrival of an aperiodic task occurrence, a global scheduler selects the processor where the incoming task obtains the best response time. This global scheduler must rely on the information provided by local aperiodic servers. Each aperiodic server should be capable of calculating response time of an incoming aperiodic task occurrence using an exact or an approximate algorithm.

2.4.2 The Global Aperiodic Scheduling

The global scheduler not only takes all the decision at run time, but also allows aperiodic tasks to migrate from one processor to another in order to improve their run times. The migration decision relies on the location of the slack gaps, i.e., the intervals where an

incoming aperiodic task can execute without jeopardizing the hard deadlines of periodic tasks. That slack gaps are provided by a slack stealing algorithm.

2.5 Motivation

The work presented in this thesis was inspired by the recent research done in the area of combined scheduling of hard and soft real-time tasks discussed above. The drawbacks of the *multimedia server* mentioned earlier and the major problem of high scheduling overhead experienced by the *0/1 and multilevel degradation algorithms* motivated the need to develop a new technique for assigning fraction of CPU time for soft tasks when considering them in hard real-time multiprocessor systems. As a result, an association policy that associates each soft task with a hard task when possible was proposed in the next chapter. Although the *aperiodic online assignment* and *global aperiodic scheduling* approaches were able to provide deterministic guarantee for hard tasks that their schedulability is not affected by the presence of soft tasks in the system, the fact that soft tasks have deadlines was completely ignored. This shortcoming motivated the work towards the development of *background and emergency combined scheduling algorithms* proposed in chapter 4.

CHAPTER 3: ASSOCIATION BASED COMBINED SCHEDULING ALGORITHM

3.1 Introduction

In this thesis we propose a novel scheduling approach for a real-time multiprocessor environment where tasks of two types soft and hard arrive dynamically. We study the problem of scheduling such dynamic tasks, where new tasks arrive or leave the system at arbitrary instances of time. Of course, it is possible that we regard each soft task as a hard real-time task and schedule it with other hard tasks. However, the cost involved in individually scheduling these tasks using dynamic scheduling algorithm would be very high. In our approach soft tasks can be accommodated in hard real-time systems by associating them to hard tasks when possible. In that case, resulting tasks from association are presented to the scheduler to be scheduled on behave of the original pairs of tasks that take part in association. Clearly, this kind of association can reduce the scheduling overhead, and provides a notion of fairness for soft tasks by assigning fraction of CPU time for them to execute. In this chapter, details about the association mechanism, rules, and algorithms are presented.

3.2 Task model

- We consider two types of tasks running on multiprocessor system with **p** identical processors. The two types are:
  - Aperiodic hard real-time tasks. Each task T_{ih} is characterized by its ready time (r_{ih}), worst-case computation time (c_{ih}), and deadline (d_{ih}).
  - Soft real-time tasks. Each task T_{is} is characterized by its ready time (r_{is}), worst-case computation time (c_{is}), and deadline (d_{is}).
- Multimedia streams are periodic soft tasks. Each stream S_i is characterized by its worst-case computation time c_i and period p_i.
- Instances of a multimedia stream S_i are considered soft tasks. Each instance T_{ij} is characterized by its ready time (r_{ij}) and worst case computation time (c_i) and deadline ($d_{ij} = r_{ij} + p_i$).

- Soft tasks are associated to hard tasks when possible.
- The resulting task T_{ia} from associating soft task T_{ks} to hard task T_{jh} is characterized by its modified ready time and deadline (r_{ia}, d_{ia}), and computation time ($c_{ia} = c_{jh} + a \times c_{ks}$) where "*a*" is the association parameter that represents the computation time percentage assigned for the soft task.
- In *full association* "*a*" is equal to 1.
- In *partial association* "*a*" is chosen between 0 and 1.
- The scheduler uses the resulting tasks from association to construct the schedule.
- Tasks are non preemptable.

3.3 Scheduler Model

In a multiprocessor system employing a dynamic scheduling algorithm, all the tasks arrive at a central processor called the scheduler, from where they are distributed to other processors in the system for execution. The communication between the scheduler and the processors is through dispatch queues. Each processor has its own dispatch queue. This organization shown in Figure 5 ensures that the processors will always find some tasks in the dispatch queues when they finish the execution of their current tasks. The scheduler runs in parallel with the processors, scheduling the newly arriving tasks, and periodically updates the dispatch queues.

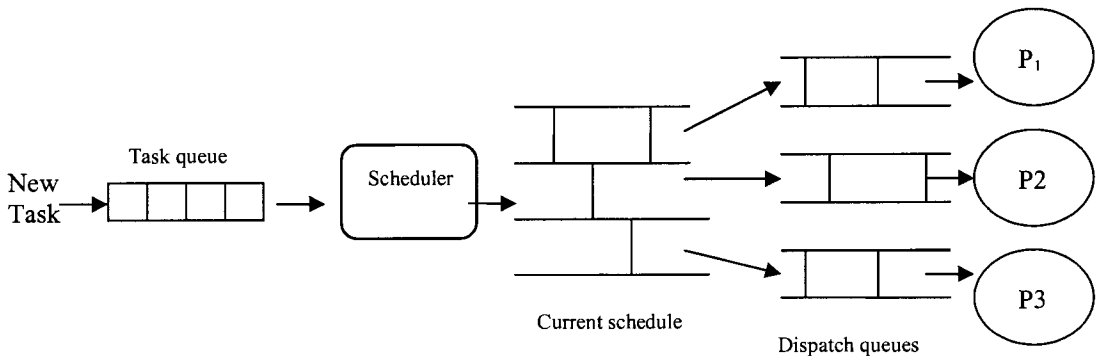


Figure 5. Scheduler Model

3.4 Association Mechanism

Considering the task model described above, association is a preprocessing phase applied to both task sets, hard and soft, to generate a merged task set to be used by the dynamic scheduler. Soft tasks are assigned fraction of CPU time by associating them to hard tasks when possible. The resulting task from associating soft task and hard task is a new task with modified ready time and deadline parameters, and with computation time equals to hard task's computation time + $a \times$ soft task's computation time. When the association parameter "*a*" is equal to 1, then complete computation time is reserved for soft tasks and the association is said to be **full**. This kind of association assumes worst-case computation time for soft tasks. If "*a*" is smaller than 1, then only fraction of CPU time is reserved for soft task execution and the association is said to be **partial**. This kind of association is based on the fact that actual task's computation time is usually less than its worst-case computation time, so there is no need for full reservation.

3.4.1 Association Rules

To illustrate the cases where association between two tasks is possible, consider two tasks, one is soft T_s with parameters (r_s, c_s, d_s), and the other is hard T_h with parameters (r_h, c_h, d_h). Let us refer to the resulting task from association by T_a with parameters (r_a, c_a, d_a). Association of the two tasks is possible in the following cases, in which the association parameter "*a*" is assumed to be equal to 1.

CASE 1: When ($r_h \leq r_s$) and ($d_s \leq d_h$) and (($r_s - r_h$) $\leq c_h$) and (($d_h - d_s$) $\leq c_h$). The resulting task T_a will have the following parameters:

- $r_a = r_h$.
- $c_a = c_h + c_s$.
- $d_a = d_h$.

In this case association is possible if the soft task is completely included within the hard task's execution interval (i.e., [r_h, d_h]) as shown in Figure 6.a. However, two conditions are imposed for the association to be correct. Firstly, the difference between ready times of the

two tasks should be less or equal to hard task's computation time (i.e. $((r_s - r_h) \leq c_h)$) to ensure that soft task will be ready as soon as hard task completes execution if T_a gets scheduled in the interval $[r_a, r_a + c_a]$, otherwise the computation time of the resulting task (c_a) should be made larger than ($c_h + c_s$) to include the hole presented between the two tasks, which is not practical option. Figure 6.b shows the need for this condition. Secondly, the difference between deadlines of the two tasks should be less than or equal to hard task's computation time (i.e. $((d_h - d_s) \leq c_h)$) to guarantee that both tasks will meet their deadlines even if T_a gets scheduled in the interval $[d_a - c_a, d_a]$. Figure 6.c shows the need for this condition to be imposed by observing that soft task will definitely miss its deadline if T_a gets scheduled in the place shown in the figure.

CASE 2: When $(r_s \leq r_h)$ and $(d_h \leq d_s)$. The resulting task T_a will have the following parameters:

- r_a = maximum $(r_s, r_h - c_s)$.
- $c_a = c_h + c_s$.
- d_a = minimum $(d_s, d_h + c_s)$.

In this case association is possible if the hard task is completely included within the soft task's execution interval. The ready time r_a and deadline d_a of the resulting task T_a are chosen in such away to give the task larger laxity. The ready time r_a can be made as early as the maximum $(r_s, r_h - c_s)$ to ensure that hard task will start execution as soon as soft task finishes if T_a gets scheduled in the place shown in Figure 7.a. The deadline d_a can be extended up to minimum $(d_s, d_h + c_s)$ to make sure that hard task will always meet its deadline even if T_a gets scheduled in the place shown in Figure 7.b.

CASE 3: When $(r_h \leq r_s)$ and $(r_s < d_h)$ and $(d_h \leq d_s)$ and $((r_s - r_h) \leq c_h)$. The resulting task T_a will have the following parameters:

- $r_a = r_h$.
- $c_a = c_s + c_h$.
- d_a = minimum $(d_s, d_h + c_s)$.

In this case association is possible if the execution interval of the soft task overlaps partially with that of the hard task as shown in Figure 8. The same condition discussed in CASE 1 regarding the difference between the two ready times is imposed here for the same reason. Also the deadline d_a is extended similar to what presented in CASE 2.

CASE 4: When ($r_s \leq r_h$) and ($r_h < d_s$) and ($d_s \leq d_h$) and ((d_h-d_s) $\leq c_h$). The resulting task will have the following parameters:

- r_a = maximum (r_s, r_h - c_s).
- $c_a = c_h + c_s$.
- $d_a = d_h$.

In this case association is possible if the execution interval of the soft task overlaps partially with that of the hard task as shown in Figure 9. The same condition discussed in CASE 1 regarding the difference between the two deadline times, is imposed here for the same reason. Also the ready time r_a is extended similar to what presented in CASE 2.

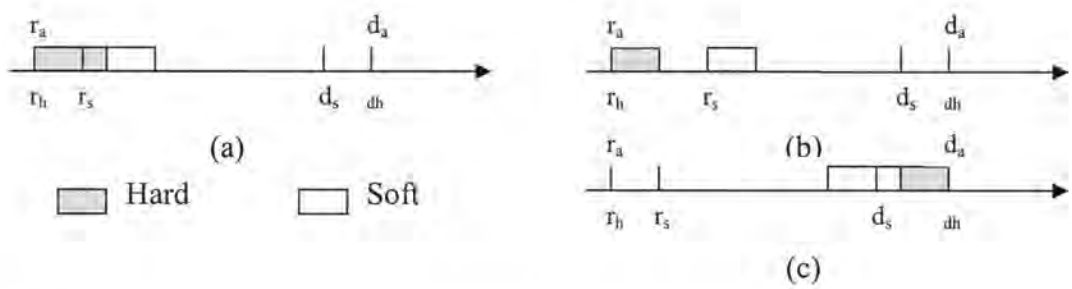


Figure 6. Case1 (a) Correct association. (b) Incorrect association because (r_s - r_h) >c_h. (c) Incorrect association because (d_h – d_s) > c_h.

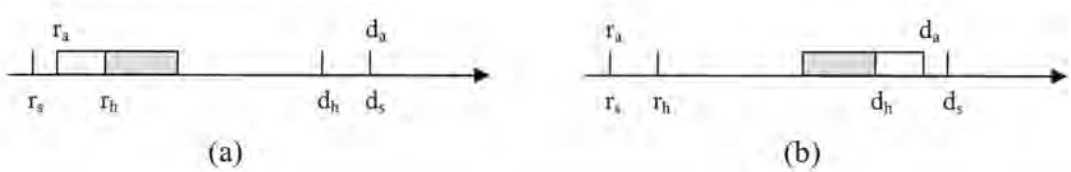


Figure 7. Case2 (a) The minimum ready time that can be assigned to T_a. (b) The maximum deadline that can be assigned to T_a.

Figure 8. Case3

Figure 9. Case4

3.4.2 Association Algorithms

To perform association between soft and hard real-time tasks two algorithms were developed. The first algorithm called the ***Exact Association Algorithm*** uses a brute force method to perform association. This algorithm represents a baseline and is not expected to be used in practice because of its high time complexity. The second algorithm called the ***Approximate Association algorithm*** uses a strategy based on deadline ordering. It has lower time complexity and performs well under different scenarios.

3.4.2.1 Exact Association Algorithm

As shown in Figure 10 this algorithm takes hard real-time tasks set **H** of size n, and soft real-time tasks set **S** of size m, as an input. This algorithm considers soft tasks one by one. For each soft task, a temporary associations set is used. Starting by an empty temporary associations set, hard tasks are examined one by one for possibility of association based on the association rules described earlier. If association can be made, a new temporary task is constructed and added to the set of temporary associations of the soft task. After checking all hard tasks, one of the temporary tasks is chosen either *randomly* or based on the *laxity*. The chosen task is added to the merged task set, and the two tasks involved in association are removed from their original task sets to avoid choosing them for association by tasks not associated yet. This process is repeated for all other soft tasks. At the end all hard and soft tasks remained without association are added to the set of merged tasks to be scheduled individually. Clearly, this algorithm depends on brute force method to find the best

association for each soft task. Its high time complexity *O (nm)* makes it unattractive to be used in real life. This motivated the need to develop the *Approximate Association Algorithm* described below.

1. ***Exact Association Algorithm*:**
2. Input: hard real-time tasks set H= [T_{1h}...T_{nh}]. Soft real-time tasks set S= [T_{1s}...T_{ms}].
3. Output: Merged Tasks set.
4. i =1;
5. while (i ≤m)
6. begin
    a. j = 1;
    b. temporary associations set = {};
    c. while (j ≤n)
    d. begin
        i. if (H[j] , S [i]) can be associated
            1. construct temporary association task T_{ij};
            2. Add Tij to the set of temporary associations;
        ii. j = j +1;
    e. end
    f. if temporary associations set is not empty
        i. choose on of the temporary associations *randomly* or based on the *laxity*;
        ii. add the chosen task T_{ik} to the merged task set;
        iii. remove S[i] and H[k] from their sets;
        iv. i =i + 1;
7. end
8. add remaining tasks to the merged task set.
9. end

Figure 10. Exact Association Algorithm

3.4.2.2 Approximate Association Algorithm

As shown in Figure 11, this algorithm uses different strategy to associate tasks to each other. It takes a task set of size (n + m) as an input. It starts by ordering tasks, soft and hard, based on their deadlines. The algorithm then works on two successive tasks each time; if the two tasks are of the same type then no association can be done and the next pair of tasks is considered. If the two tasks are of different types then they will be associated according to the association rules mentioned earlier. The resulting task is inserted in the merged task set, and the two tasks involved in association are removed from the original task set. When the

end of task set is reached, the remaining tasks, soft and hard are inserted in the merged task set. The time complexity of this algorithm is $O(n + m)$.

1. ***Approximate Association Algorithm:***
2. Input: a set of real time tasks T= $\{T_1, T_2 \ldots Tn_{+m}\}$. Task T_i can be of either types, hard or soft.
3. Output: Merged Tasks set.
4. Tasks are ordered in non-decreasing order of deadline
5. index1 = 1
6. index2 = 2
7. While(index2 ≤n + m)
8. begin
    a. If T_{index1} and T_{index2} are of the same type
        i. Index1 = index2
        ii. Index2 = index2 + 1
    b. else
        i. Associate (T_{index1}, T_{index2})
        ii. Add resulting task to the set of merged tasks
        iii. Remove T_{index1}, T_{index2} from the original task set
        iv. index1=index1+2
        v. index2=index2+2
9. end
10. Add remaining tasks to the set of merged tasks
11. end

Figure 11. Approximate Association Algorithm

3.4.3 Association Example

To illustrate the working of *Exact* and *Approximate* Association Algorithms we use the hard and soft task sets shown in tables 1 and 2 respectively. In this example we use T_{ih} to refer to the i^{th} hard task, T_{js} to refer to the j^{th} soft task, and $T_{j,i}$ to refer to the resulting task from associating T_{js} to T_{ih}. (r, c, d) notation is used to refer to the task's ready time, computation time, and deadline respectively. Table 3 shows how exact association works. For each soft task in the first column, the set of temporary associations are shown in the second column. Then the temporary association with the largest laxity is chosen. At the end hard tasks remained without association (T_{4h}, T_{6h}) are added. Based on the information presented in this table, the merged tasks set will contain the following tasks $\{T_{1,2}, T_{2,1}, T_{3,3}, T_{4,5}, T_{5,7}, T_{4h}, T_{6h}\}$. In approximate association all tasks are ordered based on their deadlines

as shown in table. 4, then two successive tasks are considered each time. When the two tasks are of different types the association is made according to the rules specified earlier, otherwise no association can be made. Tasks resulting from association are shown in column 5 of the table. At the end all tasks remained without association are added. This include (T_{1h}, T_{6h}, T_{7h} , and T_{1s}). Of course it is possible to reject soft tasks remained without association. The merged tasks set will contain the following tasks {$T_{2,2}$, $T_{3,3}$, $T_{4,5}$, $T_{5,4}$, T_{1h},T_{6h}, T_{7h} ,T_{1s}}. This example shows that by using approximate association few soft tasks may remain without association. Such tasks may be rejected at all if the aim is to keep the total number of tasks equals to the number of hard tasks. Otherwise some or all of these tasks may be considered.

Table 1. Hard Real-Time Tasks Set

TASK ID	READY TIME	COMP TIME	DEAD-LINE
1h	0	11	49
2h	0	15	50
3h	11	12	60
4h	15	20	65
5h	24	8	63
6h	33	14	82
7h	35	10	80

Table 2. Soft Real-Time Tasks Set

TASK ID	READY TIME	COMP TIME	DEAD-LINE
1s	0	8	35
2s	5	13	55
3s	12	15	61
4s	18	9	61
5s	27	10	70

Table 3. Working of Exact Association Algorithm

TASK ID	SET OF TEM PORARY ASSOCIATIONS $T_{j,i}$ (r,c,d)	LAXITY	CHOSEN ASSOCIATION $T_{j,i}$
1s	$T_{1,2}$(0,23,50)	27	$T_{1,2}$(0,23,50)
2s	$T_{2,1}$(0,24,55) $T_{2,3}$(5,25,60) $T_{2,4}$(5,33,65) $T_{2,5}$(16,21,63)	31 30 27 26	$T_{2,1}$(0,24,55)
3s	$T_{3,3}$(11,27,61) $T_{3,4}$(12,35,65)	23 18	$T_{3,3}$(11,27,61)
4s	$T_{4,4}$(15, 29,65) $T_{4,5}$(18,17, 63)	21 28	$T_{4,5}$(18,17, 63)
5s	$T_{5,6}$(27,24,82) $T_{5,7}$(27,20,80)	31 33	$T_{5,7}$(27,20,80)

Table 4. Working of Approximate Association Algorithm

TASK ID	READY TIME	COMP TIME	DEADLINE	ASSOCIATION $T_{i,i}$ (r,c,d)
1s	0	8	35	Can't be done because 49-35 > 11 (case 1)
1h	0	11	49	
2h	0	15	50	$T_{2,2}$ (0,28,55)
2s	5	13	55	
3h	11	12	60	$T_{3,3}$ (11,27,61)
3s	12	15	61	
4s	18	9	61	$T_{4,5}$ (18,17,63)
5h	24	8	63	
4h	15	20	65	$T_{5,4}$ (15,30,70)
5s	27	10	70	
7h	35	10	80	
6h	33	14	82	

3.5 Performance Studies

3.5.1 Simulation Overview

We have conducted extensive simulation study to evaluate the performance of our proposed schemes and algorithms. This section presents the task generation, simulation method, and experimental results. Performance evaluation of *Exact Association Algorithm* and *Approximate Association Algorithm* is presented and compared to the performance of *multimedia server.*

3.5.2 Task Generation

The hard real-time task set generated by this method is by itself feasible. That is, in the absence of the soft tasks, an optimal scheduler can find a schedule for the task set. The task set is generated according to the following input parameters:

- The minimum processing time of tasks, Min_C.
- The maximum processing time of tasks, Max_C.
- The minimum deadline of tasks, Min_D.
- The maximum deadline of tasks, Max_D.
- Number of processors considered for simulation, P.
- Number of resources considered for simulation, R.
- Probability that a task uses a resource, Use_P.

- Probability that a task uses a resource in shred mode, Share_P.
- The schedule length, L.

General rules

1. The generated task's processing time is randomly chosen using a uniform distribution between the minimum processing time (MIN_C) and maximum processing time (MAX_C).
2. The deadline of each task is randomly chosen between (finish time of the task + minimum deadline Min_D) and (finish time of the task + maximum deadline Max_D).

The schedule created by this task set generator is in the form of a matrix M which has r columns and L rows. Each column represents a resource and each row represents a time unit. The task set generator starts with an empty matrix. It then generates a task by selecting one of the P processors with the earliest available time and then requests the resources according to the probabilities specified in the generation parameters. The earliest available time of the selected processor becomes equal to the finish time of the generated task. The task set generator then marks on the matrix that the processor and resources required by the task are reserved for a number of time units equal to the task's computation time. The task set generator generates tasks until the remaining unused time for each processor, up to L, is smaller than the minimum processing time of a task, which means that no more tasks can be generated to use the processors.

3.5.3 Multimedia Stream Generation

Multimedia streams used in our experiments are generated according to the following parameters:

- Baseline multimedia computation time, c_b.
- Baseline multimedia period, P_b.
- Period increase ratio, r.
- Number of multimedia streams, m.

The baseline multimedia stream period P_b is the smallest period of all generated streams. The periods of other streams are made larger than P_b by multiples of the period increase ratio r. That is for any stream S_i the period $P_i = (1 + (i - 1) \times r) \times P_b$. Also the baseline multimedia stream is assigned a computation time equal to c_b, while other streams are assigned computation time larger than the baseline multimedia computation time c_b by multiples of the period increase ratio r. That is for any stream S_i the computation time $c_i = (1 + (i - 1) \times r) \times c_b$. By this method we can generate any number of streams with different periods and different computation times, such that all streams are related to one baseline stream. This enables us to study the effect of multimedia load by only changing the parameters of the base line stream.

3.5.4 Simulation Method

In our experiments we evaluate our mechanism and algorithms by comparing it to the ***proportional multimedia server***. *Proportional multimedia server* is used here because it has better performance than the individual allocation as described in [4]. In association, stream instances are associated to hard tasks using the association algorithms discussed previously. In multimedia server, the streams are mapped onto the multimedia server. All stream instances whose start times are before the latest deadline of the hard real-time task set are added to the task set. The resulting task sets are then scheduled using myopic algorithm [12]. For each run, 400 task sets were generated; the performance metric reported for certain set of parameters is the average of five runs with the same parameters. Table 5 shows the hard real-time task parameters used throughout our experiments. Table 6 shows the parameters used to generate multimedia streams.

In the first set of experiments we are interested in whether all hard real-time tasks and multimedia stream instances can meet their deadlines. Therefore, the success ratio **SR** defined as the ratio of total number of task sets found schedulable to the total number of task sets considered for scheduling, is used as our primary performance metric. In the second set of experiments we are more interested to know what is the value of association parameter "*a*" that gives better performance for soft tasks while all hard tasks are guaranteed. So, the

guarantee ratio **GR** defined as the ratio of total number of tasks found schedulable to the total number of tasks considered for scheduling, is used as another performance metric.

Table 5. Hard Real-Time Tasks set Parameters

Parameter	Value
Min_C	10
Max_C	30
Min_D	30
Max_D	60
Use_P	0.7
Share_P	0.2
R	2
P	3
L	300

Table 6. Multimedia Streams Parameters

Parameter	Value
P_b	33
C_b	2
r	0.2
m	5

3.5.5 Simulation Results

3.5.5.1 Random Based Versus Laxity Based Association

As discussed before, exact association algorithm finds all association possibilities for each soft task, and then one of the associations is to be chosen either *randomly* or based on the *laxity* of the resulting task. Here, we compare the performance of the exact association algorithm in two cases: when choosing associations randomly (Random_based) and when choosing associations based on the laxity (Laxity_based). Figure 12 shows the results of the comparison. It can be observed that the success ratio achieved using both schemes keeps decreasing as the multimedia computation time increases. This is because the increased multimedia computation time leaves less CPU time for the hard tasks, thus the tightness of scheduling increases. Laxity based association gives better success ratio than the random based because it selects the association in such a way to increase the laxity of the resulting tasks. In the following experiments we used *laxity based* for the exact association algorithm.

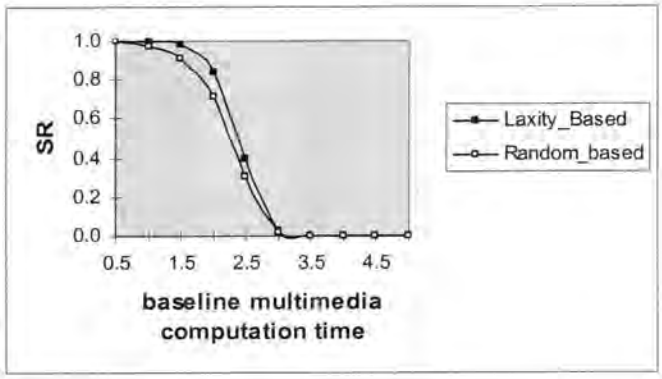


Figure 12. Comparison between Random_based and Laxity_based Exact Association Algorithms

3.5.5.2 Effect of Baseline Multimedia Computation Time

Figure 13 shows the effect of baseline multimedia computation time on the success ratio. The simulation results for the multimedia server and the two association algorithms *exact* and *approximate* are presented. It can be observed that the success ratio achieved using both schemes keeps decreasing as the multimedia computation time increases. This is because the increased multimedia computation time leaves less CPU time for the hard tasks, thus the overall schedulability decreases. However the results show that the association mechanism works better than the multimedia server. For example when the multimedia computation time is 2 time units, the success ratio achieved by the multimedia server scheme is only 42%, while it reaches 80% when using the exact association and 75% when using the approximate association. This can be explained by recalling that multimedia server instances become very tight as their computation time increases, and this leaves less CPU time for hard real-time tasks. On the other hand the association mechanism tries to make the associated tasks as relaxed as possible by flexible deadline and ready time assignments. Based on these results we conclude that association mechanism outperforms multimedia server scheme by providing higher success ratio.

3.5.5.3 Effect of Changing the Size of Computation Time of Hard Tasks

In Figure 14 the performance of the two schemes (multimedia server and association) is investigated when the execution time of hard real-time tasks is increased. In this experiment

the multimedia baseline computation time was fixed to 2 time units, Min_C was fixed to 10, while Max_C was increased in steps of 5 from 15 to 50 as shown in the figure. The results indicate that the success ratio keeps decreasing as the size of computation time of hard tasks increases. This behavior is expected because the scheduler can not find space for many hard tasks when their sizes get larger. It is clear that success ratio is affected significantly when using the multimedia server, where lower success ratio is obtained by increasing the execution time of hard real-time tasks. This can be explained by observing that hard tasks with larger execution times will find it difficult to fit between server instances, and thus the overall schedulability will decrease. On the other hand, a slight difference is noticed when exact association algorithm is used, however clear decrease in performance is noticed when using approximate association algorithm. This is because exact algorithm considers laxity of the resulting task when making association, while approximate algorithm does not. This experiment emphasizes that association policy outperforms multimedia server under different circumstances.

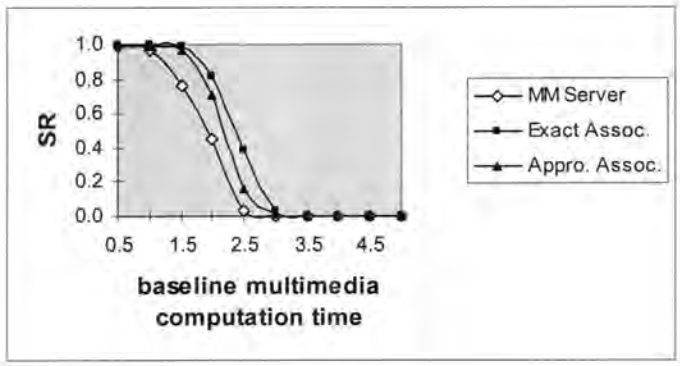


Figure 13. Effect of Baseline Multimedia Computation Time

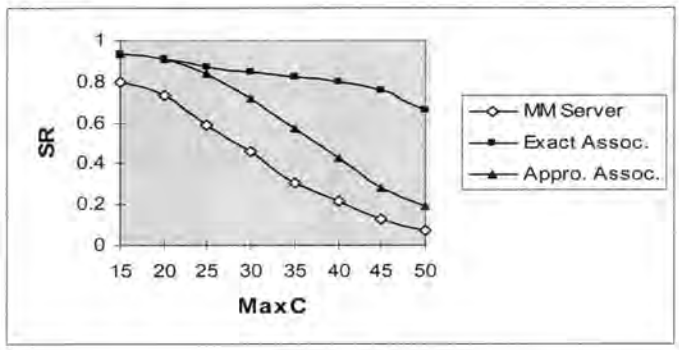


Figure 14. Effect of Hard Real-Time Tasks Size

3.5.5.4 Effect of Laxity

Figure 15 shows the effect of average hard tasks' laxity on the success ratio. In this experiment the multimedia baseline computation time was fixed to 2 time units, and the average hard tasks' laxity was increased from 30 to 80 time unites by changing Min_D and Max_D parameters used in task generation. It can be observed that association mechanism outperforms multimedia server when the average laxity is smaller than 50 time units. For example when the average laxity is 45 the success ratio obtained by exact association is 98% and by approximate association is 85%, while it is 64% if multimedia server is used. On the other hand, when the average laxity of hard tasks keeps increasing beyond 50, the situation is different. Success ratio keeps increasing significantly for multimedia server, while it drops fast toward zero for exact and approximate association. This can be explained by recalling that periods of the multimedia streams used in our simulation is in the range from 30 to 60, which means that multimedia stream instances would have better chance for association if the hard tasks have average laxity within the same range. Increasing the average laxity of hard tasks makes it difficult to satisfy the association rules mentioned earlier. While for multimedia server, increasing the average laxity of hard tasks makes the effect of server instance on hard tasks schedulability negligible. We can conclude from these results that association mechanism should be considered only when the average laxity of hard tasks is within the range of multimedia streams periods.

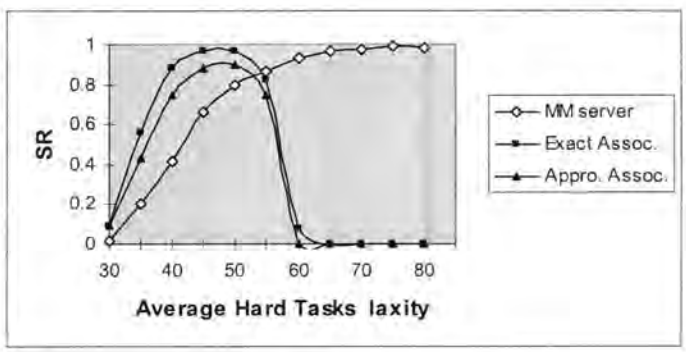


Figure 15. Effect of Laxity

3.5.5.5 Effect of the Association Parameter "a"

Figure 16 shows the guarantee ratio for each type of tasks, hard and soft, measured at run time by considering the actual execution time of each task rather than its worst-case execution. Exact and approximate association algorithms were used in this experiment. Two important things are noticed here. First, the guarantee ratio for soft tasks keeps increasing as parameter "a" increased from 0 to 0.7, then it starts decreasing fro both hard and soft tasks when "a" is larger than 0.7. This can be explained by recalling that a small value of "a" means small fraction of CPU time is reserved for soft tasks. Increasing "a" means more time is reserved and hence the chance is better for soft tasks. However if this increase goes beyond certain limit, the size of resulting tasks from association becomes large enough such that it would be difficult for them to fit in the schedule. Rejecting one associative task is equivalent to rejecting one hard task and one soft task at the same time, and this explains the decrease in the guarantee ratio for both types of tasks when "a" increases. Second, the exact algorithm provides better guarantee ratio for soft tasks, while approximate algorithm provides better guarantee ratio for hard tasks. This is because in exact algorithm more soft tasks are involved in association than in approximate algorithm.

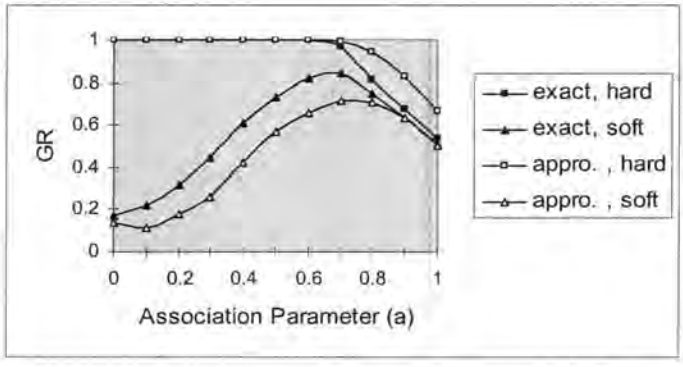


Figure 16. Effect of Association Parameter "a"

CHAPTER 4: COMBINED SCHEDULING WITH HARD TASKS GUARANTEE

4.1 Introduction

In the previous chapter we proposed the association mechanism for combined scheduling of hard and soft tasks in multiprocessor systems. The objective was to reduce the scheduling cost regardless the fact that soft tasks may affect the schedulability of hard tasks. In this chapter, we focus on developing scheduling algorithms that guarantee no hard task will miss its deadline because of scheduling of any soft task in the system, and aim at scheduling as many soft tasks as possible. Two algorithms are proposed in this chapter to achieve this objective. The first algorithm is the *Background Based Combined Scheduling Algorithm,* which has the ability to eliminate the effect of soft tasks completely by ignoring their existence when hard tasks are considered for scheduling. Also it has the background feature by which it can introduce some soft tasks in places where hard tasks are not available. The second algorithm is the *Emergency Based Combined Scheduling Algorithm,* which has the ability to drop some of already scheduled soft tasks to fulfill the timing requirements of hard tasks in critical conditions (i.e. when such tasks cannot meet their deadlines).

4.2 Task Model and Assumptions

- We consider two types of tasks running on multiprocessor system with **p** identical processors. The two types are:
  - Aperiodic hard real-time tasks. Each task T_{ih} is characterized by its ready time (r_{ih}), worst-case computation time (c_{ih}), and deadline (d_{ih}).
  - Aperiodic soft real-time tasks. Each task T_{is} is characterized by its ready time (r_{is}), worst-case computation time (c_{is}), and deadline (d_{is}).
- Tasks are independent and do not have any resource constraints.
- Hard tasks are feasible by themselves (i.e. in the absence of soft tasks, the scheduling algorithm will be able to construct a feasible schedule for hard tasks).

- The scheduling algorithm has complete knowledge about the currently active task set, but not about any new tasks that may arrive while scheduling the current set.

4.3 Terminology and Definitions

In this section we define the terms and notations used by the proposed *Background Based* and *Emergency Based Combined Scheduling Algorithms*. Consider a processor P_j with set of tasks – that includes hard and soft - scheduled on it, we define the following terms for each algorithm.

4.3.1 Terms Used By Background Based Combined Scheduling Algorithm

Definition 1: The Hard Earliest Available Time of processor P_j, denoted as HEAT (P_j), is the earliest available time of processor P_j for hard tasks, defined as
HEAT (P_j) = finish time of the last hard task scheduled on processor P_j.

Definition 2: The Soft Earliest Available Time of processor P_j, denoted as SEAT (P_j), is the earliest available time of processor P_j for soft tasks, defined as
SEAT (P_j) = finish time of the last task scheduled on processor P_j.

Definition 3: The Earliest Start Time of task T_{ix}, (x = h for hard tasks, and x = s for soft tasks), is the earliest time when its execution can be started, defined as

EST (T_{ix}) = max (r_i, $\min_{j \in P}$ (XEAT (Pj)). (X= H for hard tasks or S for soft tasks).

Definition 4: A task T_{ix} is feasible in the schedule if EST (T_{ix}) + $c_{ix} \leq d_{ix}$.

4.3.2 Terms Used By the Emergency Based Combined Scheduling Algorithm

Definition 5: The Emergency Earliest Available Time of processor P_j, denoted as EEAT (P_j), is the earliest available time of processor P_j for hard tasks, defined as

EEAT (P_j) = finish time of the last hard task scheduled on processor P_j after being deferred toward its arrival time as much as possible, even if some soft task already scheduled on that processor are dropped.

Definition 6: The Regular Earliest Available Time of processor P_j, denoted as REAT (P_j), is the earliest available time of processor P_j, defined as
REAT (P_j) = finish time of the last task scheduled on processor P_j.

Definition 7: The Emergency Earliest Start Time of task T_i is the earliest time when its execution can be started in cases of emergency, defined as

EEST (T_i) = max (r_i, $\min_{j \in P}$ (EEAT (Pj)).

Definition 8: The Regular Earliest Start Time of task T_i is the earliest time when its execution can be started in regular situation, defined as

REST (T_i) = max (r_i, $\min_{j \in P}$ (REAT (Pj)).

Definition 9: A task T_i is feasible in the schedule based on REAT (P_j) if REST (T_i) + $c_i \leq d_i$.

Definition 10: A task T_i is feasible in the schedule based on EEAT (P_j) if EEST (T_i) + $c_i \leq d_i$.

4.3.3 Terms Used by Both Algorithms

Definition 11: A feasibility check window of size k contains k tasks from which one task is chosen to extend the current schedule.

Definition 12: The partial schedule is said to be *strongly feasible* if all tasks in the feasibility check window are feasible.

Definition 13: The partial schedule is said to be *hard feasible* if all hard tasks in the feasibility check window are feasible, and at least one soft task is not feasible.

Definition 14: The partial schedule is said to be *not feasible* if at least one hard task in the feasibility check window is not feasible.

4.4 Background Based Combined Scheduling Algorithm

This algorithm eliminates the effect of soft tasks by ignoring their existence completely when considering hard tasks for scheduling. This can lead to some hard tasks get scheduled in the place of soft tasks already scheduled. In this case, such already scheduled soft tasks are either dropped completely (**indivisible-soft**), or can be divided such that non-overlapping parts remain in their places (i.e., non-overlapping portion of the soft tasks with hard tasks), and overlapping parts are considered again for scheduling as new tasks (**divisible-soft**). It is possible to keep dividing soft tasks each time they found to overlap in the schedule with hard tasks; however the number of divisions for each soft task was limited to 2 to eliminate the huge number of tasks and hence scheduling overhead that can result due to such a dividing feature. It is clear that this strategy ensures that soft tasks will not affect the schedulability of hard tasks at all, because the scheduler accepts them only if there are no hard tasks to be scheduled in their places.

As shown in Figure 17, this algorithm works similar to myopic algorithm [12]. Tasks in the task queue are ordered in non-decreasing order of deadlines. Starting with an empty partial schedule, the algorithm determines the feasibility of the partial schedule resulting from extending the current schedule by one of the tasks in the feasibility check window. If the partial schedule is found to be strongly feasible, the algorithm computes a heuristic function for each task within the feasibility check window and extends the schedule by the task having the minimum heuristic value. Based on the type of the task, the earliest available times of the chosen processor P_j (i.e. HEAT (P_j) and SEAT (P_j)) are updated differently for hard and for soft tasks. If the chosen task is hard, then the algorithm should manage the overlapping that may occur with already scheduled soft tasks, either by rejecting such soft tasks in the case of *indivisible-soft* version of this algorithm, or by keeping non-overlapping parts of these tasks in their places, and inserting the overlapping parts as new soft tasks in the task queue to be considered later for scheduling in the case of *divisible-soft* version of the

algorithm. If the chosen task is soft, the task is scheduled and only the SEAT (P_j) is updated. If the partial schedule is found to be hard feasible, the same actions are taken by the algorithm, however non feasible soft tasks in the feasibility check window are rejected directly. If the current partial schedule is not feasible, the algorithm backtracks to the search level at which the last hard task was scheduled and extends the schedule with the hard task having the next minimum heuristic value. The feasibility check window is then moved forward to include one new task from the task queue, and the previous operation is repeated until a feasible schedule (or hard feasible) is constructed.

1. **Background Based Combined Scheduling()**
2. **Begin**
3. Tasks (in the task queue) are ordered in non-decreasing order of deadline.
4. Determine the feasibility of the current schedule with respect to the *k* tasks in the feasibility check window.
5. **CASE 1**: strongly feasible or hard feasible
    a. Reject non feasible soft tasks
    b. Compute heuristic function H_{ix} for each task, where $H_{ix} = d_{ix} + W \times EST(T_{ix})$.
    c. Extend the schedule with the task T_{ix} having the best (smallest) H value
        i. **if** (T_{ix} is hard)
            1. HEAT (P_j) = EST (T_{ih}) + c_{ih}.
            2. SEAT (P_j) = EST (T_{ih}) + c_{ih}.
            3. Manage soft tasks overlapping with T_{ih}.
        ii. **else**
            1. SEAT (P_j) = EST (T_{is}) + c_{is}.
6. **CASE 2**: not feasible
    a. Backtrack to the search level at which last hard task is scheduled
    b. Extend the schedule with the hard task having the next –best H value.
7. **Repeat** steps (3- 6) **until** a feasible or hard feasible schedule is obtained

Figure 17. Background Based Combined Scheduling Algorithm

4.5 Emergency Based Combined Scheduling Algorithm

Although *Background Based Combined Scheduling Algorithm* discussed in the previous section provides the required guarantee that soft tasks will not affect the schedulability of hard tasks, it is very pessimistic in the sense that many soft tasks get rejected simply because they are scheduled in the same time interval of other hard tasks. This drawback of *Background Based Combined Scheduling* motivated the need for new algorithms that can

provide better service for soft tasks while still preserving the guarantee for hard tasks. In this section we propose the *Emergency Based Combined Scheduling Algorithm* which achieves this goal. As the name indicates, this algorithm has an emergency feature, by which some soft tasks already scheduled can be dropped to fulfill the timing requirements of a hard task in critical condition (i.e. if it cannot meet its deadline).

As shown in Figure 18, this algorithm is similar to the *Background Based Combined Scheduling Algorithm* described earlier in the strategy used to construct the schedule.

1. **Emergency Based Combined Scheduling()**
2. **Begin**
3. Tasks (in the task queue) are ordered in nondecreasing order of deadline.
4. Determine the feasibility of the current schedule with respect to the k tasks in the feasibility check window, based on the REAT (P_j).
5. **CASE 1**: strongly feasible or Hard feasible
    a. Reject non feasible soft tasks
    b. Compute heuristic function H_{ix} for each task, where $H_{ix} = d_{ix} + W \times REST(T_{ix})$.
    c. Extend the schedule with the task T_i having the best (smallest) H value.
        i. **if** (T_i is hard)
            1. $EEAT(P_j) = EEST(T_{ih}) + c_{ih}$.
            2. $REAT(P_j) = REST(T_{ih}) + c_{ih}$.
        ii. **else**
            1. $REAT(P_j) = REST(T_{is}) + c_{is}$.
6. **CASE 2**: not feasible
    a. Determine the feasibility of the current schedule with respect to the k tasks in the feasibility check window, based on the EEAT (P_j).
    b. **CASE 21**: strongly feasible or Hard feasible
    c. Reject non feasible soft tasks
    d. Compute heuristic function H_{ix} for each hard task, where $H_{ix} = d_{ix} + W \times REST(T_{ix})$.
    e. Extend the schedule with the hard task T_{ih} having the best (smallest) H value.
        i. Defer tasks scheduled on P_j back to the EEAT (P_j)
        ii. $EEAT(P_j) = EEST(T_{ih}) + c_{ih}$.
        iii. $REAT(P_j) = REST(T_{ih}) + c_{ih}$.
    f. **CASE 22**: not feasible
        i. Backtrack to the search level at which last hard task is scheduled
        ii. Extend the schedule with the hard task having the next –best H value.
7. **Repeat** steps (3- 6) **until** a feasible or hard feasible schedule is obtained.

Figure 18. Emergency Based Combined Scheduling Algorithm

However, it is more optimistic in the sense that soft tasks are scheduled regularly with hard tasks, and they are not dropped unless they are discovered to affect the schedulability of

hard tasks considered later. In this algorithm, feasibility of the tasks in the check window is determined initially based on the regular earliest available time of all processors in the system. If the partial schedule resulting from extending the current schedule by one of the tasks in the window is strongly feasible, the algorithm computes a heuristic function for each task within the feasibility check window and extends the schedule by the task having the minimum heuristic value. Based on the type of the task, the earliest available times of the chosen processor P_j are updated differently for regular and emergency cases. If the chosen task is hard, both emergency and regular earliest available times of processor P_j (i.e. REAT (P_j) and EEAT (P_j)) are updated. If the chosen task is soft, only the REAT (P_j) is updated. If the partial schedule is found to be hard feasible, the same actions are taken by the algorithm, however non-feasible soft tasks in the feasibility check window are rejected directly. If the schedule is not feasible, the feasibility of tasks in the window is checked based on the emergency earliest available time of all processors. If the resulting partial schedule is strongly or hard feasible, then the hard task with the minimum heuristic value is chosen to extend the schedule on the processor P_j with the minimum EEAT. At this point, the last hard task scheduled on P_j should be deferred towards its ready time such that its finish time becomes equal to EEAT (P_j) to provide a room for the new hard task to fit in the schedule. In this process one or more soft tasks already scheduled on P_j must be dropped to make the necessary shift of tasks possible. This recovery step ensures that soft tasks discovered to affect the schedulability of hard tasks will be dropped. If the partial schedule is not feasible, the algorithm backtracks to the search level at which the last hard task was scheduled and extends the schedule with the hard task having the next minimum heuristic value. The feasibility check window is then moved forward to include one new task from the task queue, and the previous operation is repeated until a feasible schedule (or hard feasible) is obtained.

4.6 Scheduling Example

To illustrate the working of the proposed algorithms, consider the set of hard and soft real-time tasks shown in table 7. This set consists of 8 hard tasks (1h... 8h), and 6 soft tasks (1s... 6s). These tasks are ordered based on deadline. The input values for k, W (the constant used to compute the heuristic), and number of processors are taken as 4, 1, and 3,

respectively. Figure 19.a is the schedule constructed using the *indivisible-soft* version *of background based algorithm*. This algorithm is unable to accept more than two soft tasks (3s and 6s). Figure 19.b is the schedule constructed using the *divisible-soft* version. In this case the algorithm was able to accept one more soft task (1s) by dividing it and scheduling each division on different processor as shown in the figure. As expected, soft tasks get better schedulability by using the *emergency based combined algorithm* as shown in Figure 19.c. Here four soft tasks get scheduled.

Figure 20 shows the search tree constructed by background based combined scheduling algorithm. Each node in the search tree is represented by two boxes: the left box shows the soft earliest available time (SEAT) of processors used to extend the schedule by a new soft task, while the right box shows the hard earliest available time (HEAT) of processors used to extend the schedule by a new hard task. For example, a left box entry of (13, 11, 7) means that the minimum soft earliest available time of all processors is that of processor 3 (i.e. 7), while a corresponding right box entry of (13, 11, 0) means that hard tasks can get scheduled on processor 3 as early as time 0. Similarly, Figure 21 shows the search tree constructed by *emergency based combined scheduling algorithm*. Each node in the search tree is represented by two boxes: the left box shows the regular earliest available time (REAT) of processors used to extend the schedule by a new task in regular situation (i.e. no hard tasks are missing their deadlines), while the right box shows the emergency earliest available time (EEAT) of processors used to extend the schedule by a new hard task in emergency cases (i.e. when a hard task is found to be not feasible because of soft tasks already scheduled). For example, a left box entry of (46, 35, 37) means that the minimum regular earliest available time of all processors is that of processor 2 (i.e. 35), while a corresponding right box entry of (46, 35, 24) means that tasks scheduled on processor 3 can be deferred back towards time 24, such that a hard task in critical condition (i.e. will miss its deadline if no action is being taken) can start as early as that time. In both figures (21 and 22) the forward arcs correspond to extending the schedule by one task at a time. The label T_a (b) on a forward arc denotes that the task T_a is scheduled on processor P_b.

Table 7. An Example of Hard and Soft Real-Time Tasks

Task ID	Ready Time	Computation Time	Deadline
1s	0	8	13
1h	0	11	15
2s	0	7	15
2h	0	13	16
3h	5	9	17
4h	13	12	27
3s	12	8	29
5h	14	10	30
4s	10	12	31
6h	21	11	35
7h	21	10	37
5s	19	11	39
8h	21	14	48
6s	22	9	49

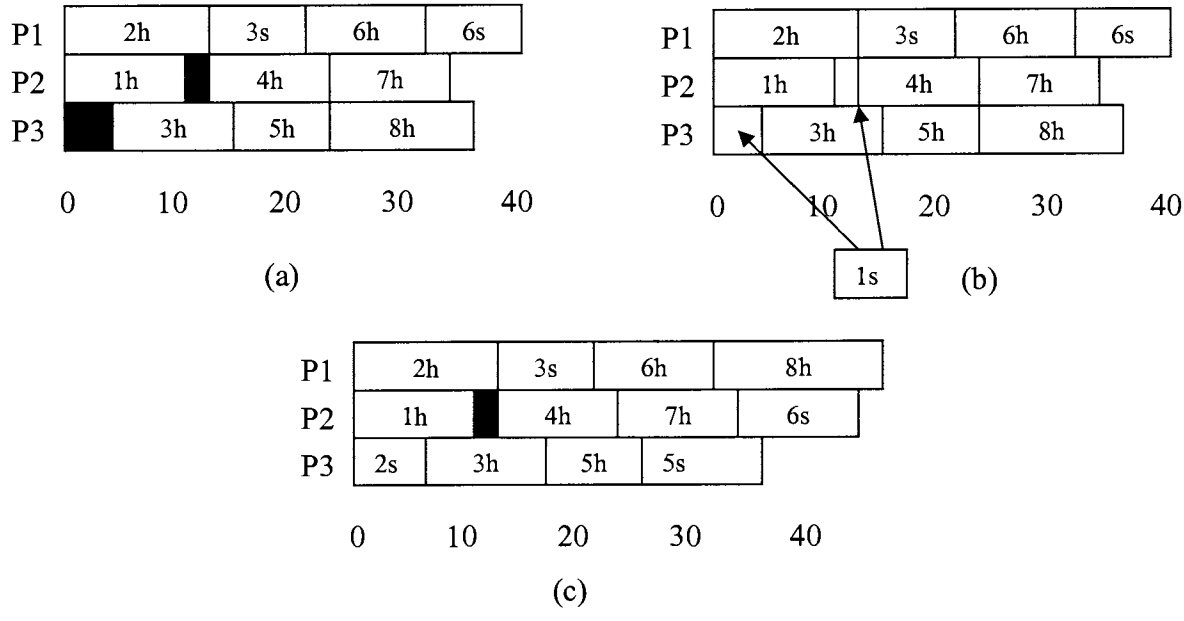


Figure 19. Schedules Constructed Using (a) indivisible-soft Background Algorithm (b) divisible-soft Background Algorithm (c) Emergency Based Algorithm.

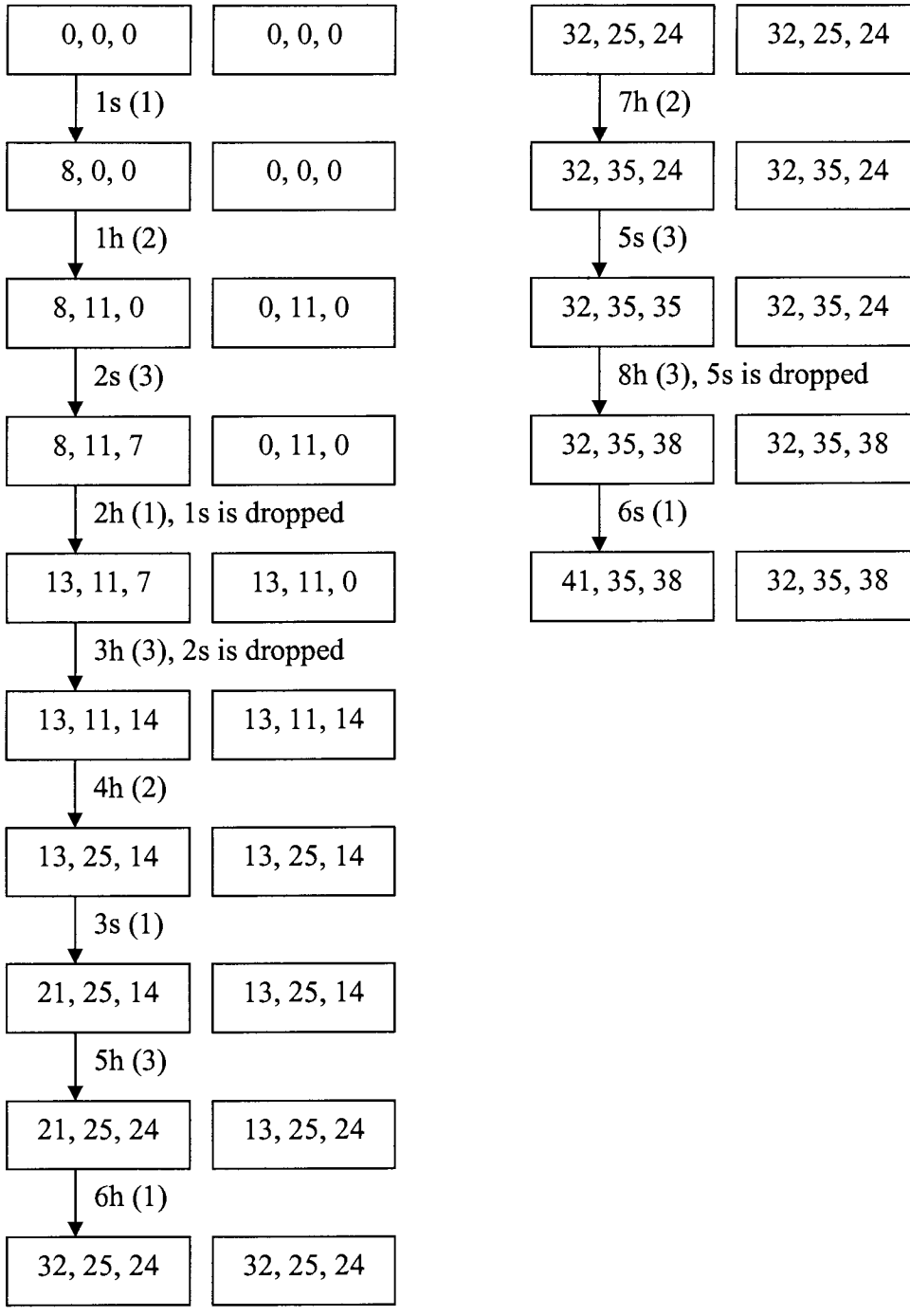


Figure 20. Search Tree of the Background Based Combined Scheduling Algorithm (indivisible-soft version)

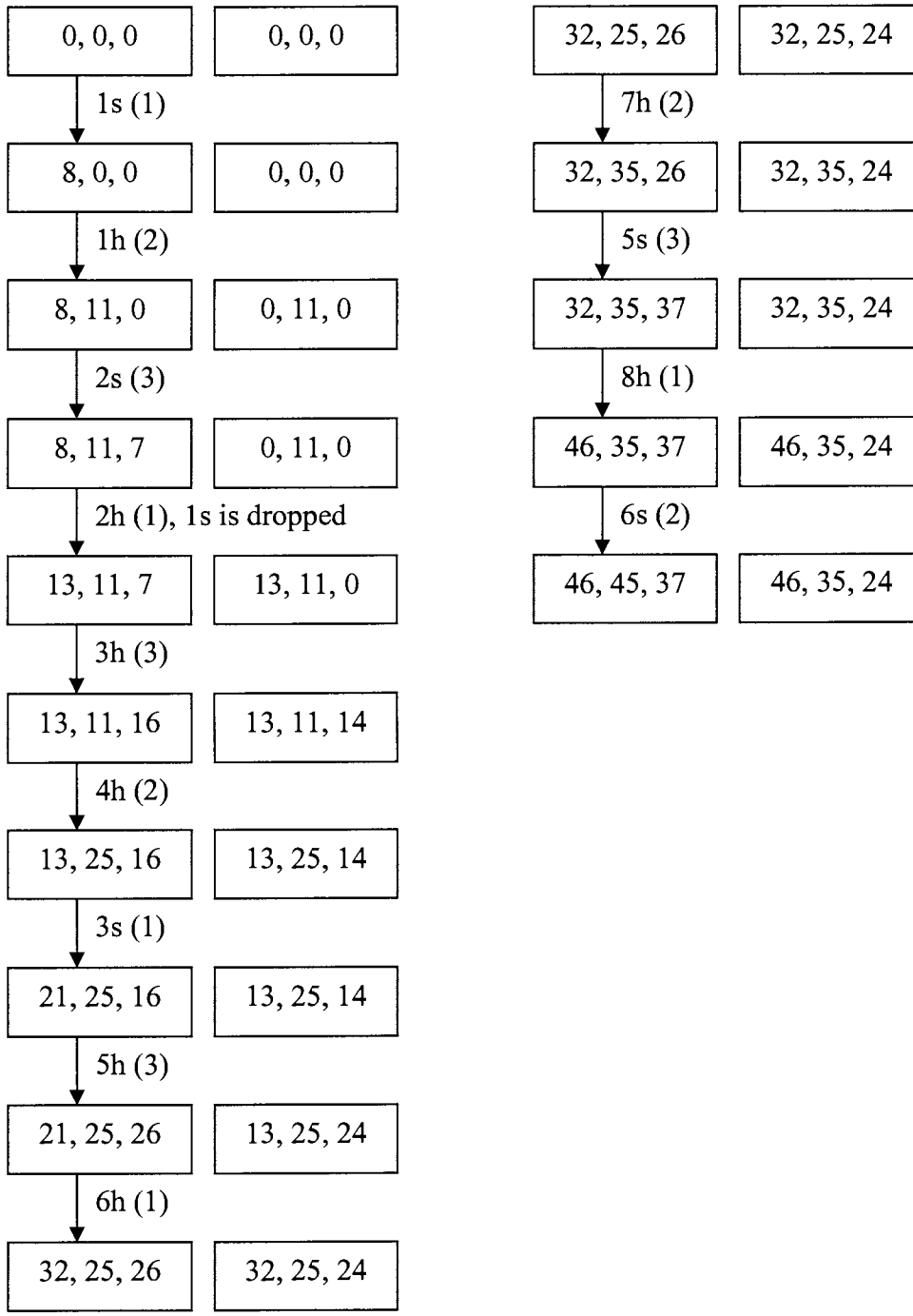


Figure 21. Search Tree of the Emergency Based Combined Scheduling Algorithm

4.7 Simulation Studies

4.7.1 Simulation Overview

We have conducted extensive simulation studies to evaluate the performance of the proposed schemes and algorithms. In the following sections we present the task generation, simulation method, and experimental results. Performance evaluation of *Background* and *Emergency Based Combined Scheduling Algorithms* is presented.

4.7.2 Task Generation

Since both task types described in the task model are aperiodic, the same task set generation method is used to generate both task sets, hard and soft. The generation process is similar to that described in section 3.5.2 and is done according to input parameters given below. A two dimensional matrix – with *time* as one dimension and *processors* as the other dimension - is used to generate a task set in such away a schedule of tasks is created by arranging these tasks in the matrix one after another. A task is generated by selecting one of the processors with the earliest available time. The earliest available time of the selected processor becomes equal to the finish time of the generated task plus a certain *distance* given as an input parameter. For example, if the generated task on processor 1 starts at time 20 and finishes at time 33 (i.e. its computation time is 13), then the earliest available time of processor 1 becomes equal to 33 + *distance*, which means that the next task generated on the same processor will have a start time of 33 + distance. The use of *distance* parameter in the task generation process enables us to provide different hard and soft loads. It also makes it possible to generate tasks at separate times, and this is important especially when we evaluate the *background combined scheduling algorithm* in which soft tasks can not fit in the schedule unless there are no hard tasks within some intervals. It is worth mentioning that each task set is generated separately according to the following parameters.

- The minimum processing times of (hard and soft tasks), HMin_C and SMin_C.
- The maximum processing times of (hard and soft tasks), HMax_C and SMax_C.
- The minimum deadlines of (hard and soft tasks), HMin_D and SMin_D.

- The maximum deadlines of (hard and soft tasks), HMax_D and SMax_D.
- The distance between tasks generated on the same processor, Hdistance (for hard tasks), Sdistance (for soft tasks).
- Number of processors considered for simulation, P.
- The schedule length, L.

4.7.3 Simulation Method

Since the proposed algorithms were designed to guarantee hard tasks by default, the performance metric used to evaluate theses algorithms was the guarantee ratio (**GR**) provided for soft tasks. The two algorithms were compared under different situations. Through out the experiments we used 4 processors system. The input parameters used to generate hard and soft tasks are shown in table 8.

Table 8. Parameters used to generate hard and soft tasks

Parameter	value	Parameter	Value
HMin_C	10	SMin_C	10
HMax_C	30	SMax_C	30
HMin_D	60	SMin_D	60
HMax_D	90	SMax_D	90
Hdistance	15	Sdistance	15

4.7.4 Simulation Results

In this section we present a comparison study between the proposed algorithms. The effect of distance of hard tasks (Hdistance), distance of soft tasks (Sdistance), and the average hard tasks size is investigated in the following subsections.

4.7.4.1 Effect of Distance of Hard Tasks Parameter (Hdistance)

Figure 22 shows the effect of the distance between hard tasks generated on the same processor (Hdistance) on the guarantee ratio of soft tasks. The simulation results of the

background (soft-indivisible and soft-divisible) combined scheduling algorithm, and the *emergency based combined scheduling algorithm* are presented. In this experiment the Sdistance parameter was fixed to 15 units. It is clear that the guarantee ratio achieved by all algorithms keeps increasing by increasing the Hdistance parameter. Increasing this parameter means that fewer hard tasks with larger separation times are generated. This provides better chance for soft tasks to get scheduled. As expected, the divisible-soft version of the *background based algorithm* performs better than the indivisible-soft version. This is because dividing a soft task when found to overlap with a hard task and considering each division separately gives better chance for such task to be scheduled. The *emergency based algorithm* significantly outperforms the background based algorithms. For example, when the Hdistance is 20 the guarantee ratio reaches 60% using the *emergency based*, while it is only 20% using the divisible-soft *background based algorithm*. This significant difference is due to the low level service provided to soft tasks by the *background based algorithm*.

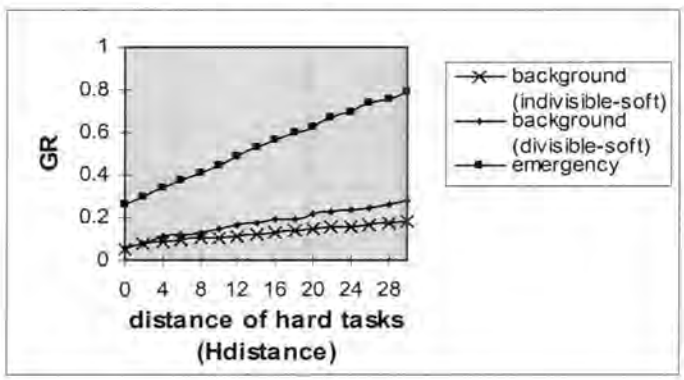


Figure 22. Effect of Distance of Hard Tasks Parameter (Hdistance).

4.7.4.2 Effect of Distance of Soft Tasks Parameter (Sdistance)

Figure 23 shows the effect of the distance of soft tasks parameter (Sdistance) on the guarantee ratio of soft tasks. The simulation results of the *background* (soft-indivisible and soft divisible) *combined scheduling algorithm*, and the *emergency based combined scheduling algorithm* are presented. In this experiment Hdistance parameter was fixed to 15 units while that of soft tasks was changed from 0 to 30. The results suggest that the guarantee ratio achieved using *background based algorithms* remains almost constant and is not affected significantly by increasing the Sdistance parameter. This is due to the fact that soft

tasks schedulability is mainly affected by the presence of hard tasks when scheduling them in the background. In contrast, the guarantee ratio achieved by *emergency based combined scheduling algorithm* keeps increasing as the Sdistance parameter increases, because soft tasks competition on resources becomes lower and hence their schedulability becomes higher.

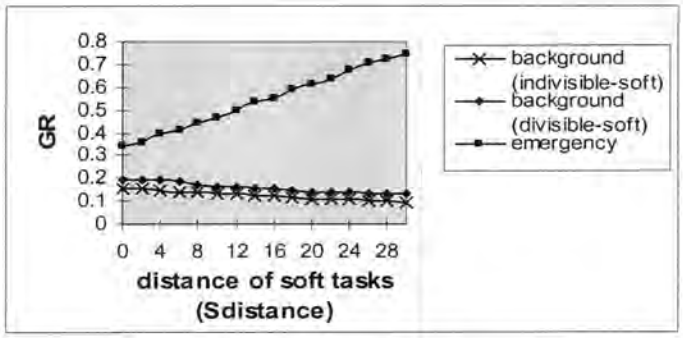


Figure 23. Effect of Distance of Soft Tasks Parameter (Sdistance)

4.7.4.3 Effect of the Average Size of Hard Tasks

Figure 24 shows the effect of average size of hard tasks on the guarantee ratio provided to soft tasks. In this experiment HMinC was fixed to 10, while HMaxC was changed from 15 to 35 in steps of 5. Increasing the average computation of hard tasks makes it difficult for them to fit in the schedule in the presence of soft tasks. This explains the decrease in performance experienced by *emergency based scheduling algorithm*. As HMaxC increases more soft tasks have to be rejected to meet the deadlines of the hard tasks. The results obtained when using *background based algorithms* emphasize that the schedulability of soft tasks is only affected by the presence of hard tasks rather than any other parameter.

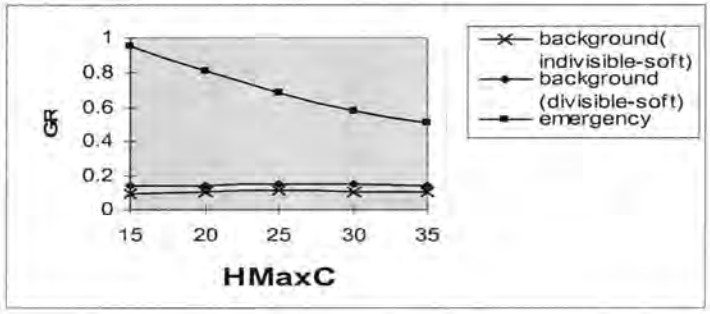


Figure 24. Effect of average size of hard tasks

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

In this thesis, we addressed the issue of combined scheduling of hard and soft real-time tasks in multiprocessor systems with two main objectives. The first objective was to avoid high scheduling overhead caused by the large number of tasks in such systems, and to provide a notion of fairness for soft tasks by assigning fraction of CPU time for them to execute. We proposed an association scheme that takes advantage from the fact of time overlapping between hard and soft tasks, such that each soft task is associated to a hard task when possible, and the pair of tasks is presented to the scheduler as one single task. We systematically described how association could be performed. Two association algorithms were developed; one is *exact* with high time complexity that makes it unattractive to be used in practice, and the other is *approximate* with lower time complexity. We have also shown how the proposed scheme can be adapted for integrated scheduling of multimedia streams and hard real-time tasks. The second objective was to provide a guarantee that hard tasks schedulability is not affected by the presence of soft tasks in the same system, with a secondary goal to accept as many soft tasks as possible. Two combined scheduling algorithms were proposed in this thesis to achieve this objective, one is the *background based combined scheduling algorithm*, which has the ability to eliminate the effect of soft tasks completely by ignoring their existence when hard tasks are considered for scheduling. Also it has the background feature by which it can introduce some soft tasks in places where hard tasks are not available. The other algorithm is the *emergency based combined scheduling algorithm*, which has the ability to drop some of the already scheduled soft tasks to fulfill the timing requirements of hard tasks in critical conditions (i.e. when such tasks cannot meet their deadlines). Simulation results indicate that emergency based combined scheduling algorithm significantly outperforms the background combined scheduling algorithm.

From the simulation studies performed to compare *association mechanism* – implemented using *exact* and *approximate* algorithms – with well-known *multimedia server mechanism*, we can draw the following conclusions:

- The success ratio achieved using both schemes keeps decreasing as the multimedia computation time increases. This is because the increased multimedia computation time leaves less CPU time for the hard tasks, thus the overall schedulability decreases.
- The results indicate that the success ratio achieved by both schemes keeps decreasing as the size of computation time of hard tasks increases. This behavior is expected because the scheduler cannot find space for many hard tasks when their sizes get larger.
- By increasing the laxity of the hard tasks, the success ratio obtained by the *multimedia server scheme* keeps increasing, because the effect of server instances on hard tasks becomes negligible as their laxity increases. The behavior experienced by the *association mechanism* is quite different; the success ratio keeps increasing up to certain limit, and then starts decreasing steeply towards zero as the laxity gets larger.
- *Association mechanism* in general outperforms *multimedia server mechanism* under different circumstances. However this mechanism is found to be effective only when the average laxity of hard tasks is within the range of periods of multimedia streams in the system.
- In all conducted experiments the *exact Association* showed better performance than *approximate association* when the success ratio metric was used.
- Association parameter "a" affects the guarantee ratio of both types of tasks. This parameter can be adjusted to provide the desired guarantee for hard and soft tasks.

Limitations of the *Association Mechanism*

- Although this mechanism provides notion of fairness for soft tasks, it does not eliminate their effect on the schedulability of hard tasks. As a result, no guarantee can be given for hard tasks.

- This mechanism is only effective when the average laxity of hard tasks is within the range of periods of the multimedia streams.
- The high time complexity of exact association algorithm makes it unattractive to be used in practice.

From the simulation studies performed to compare *background based* and *emergency based combined scheduling algorithms*, we can draw the following conclusions:

- Emergency based algorithm significantly outperforms background based algorithm.
- The distance of hard tasks parameter (Hdistance) is the main factor that affects the schedulability of soft tasks when using the background based algorithm.
- divisible-soft version of the background based algorithm slightly outperforms the indivisible-soft version of the same algorithm.
- Increasing the average size of hard tasks reduces the performance of the emergency based algorithm.

Limitations of *Background* and *Emergency Based Combined Scheduling Algorithms*

- Tasks are independent and without resource constraints.
- Background based algorithm provides low schedulability for soft tasks.

Future Work

Future research directions will focus on the following aspects:

1) To investigate new methods to achieve the two objectives presented in this thesis at the same time. Clearly, this is not a trivial problem, and it requires the consideration of several parameters at the same time.
2) To develop an adaptive algorithm that has the ability to adjust the association parameter "a", such that an optimal or near optimal performance can be obtained.
3) To develop preemptive scheduling algorithms that achieves the given objectives.

REFERENCES

[1] L. Abeni, G. Buttazzo, "Integrating Multimedia Applications in Hard Real-Time Systems," *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pp. 3-13, IEEE Computer Society Press, 1998.

[2] M. Anita, G. Manimaran, and Siva Ram Murthy, "Integrated Scheduling of Hard and QoS Degradable Real-Time Tasks in Multiprocessor Systems," *Proceedings of IEEE International Conference on Real-Time Systems and Applications*, Hiroshima, Japan, 1998.

[3] G.C. Buttazzo and F. Sensini, "Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Environments," *IEEE Transactions on Computers*. vol. 48, no. 10, pp. 1035-1051, October 1999.

[4] H. Kaneko, J.A. Stankovic, S. Sen, and K. Ramamritham, "Integrated Scheduling of Multimedia and Hard Real-Time Tasks," *Proceedings of Real-Time Systems Symposium*, pp. 206-217, 1996.

[5] S. Lee, H. Kim and J. Lee, "A Soft Aperiodic Task Scheduling in Dynamic-Priority Systems," *Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications*, IEEE 1995.

[6] J.P. Lehoczky, L. Sha, and J.k. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments." *Proceedings of Real-Time Systems Symposium,* pp. 261-270, 1987.

[7] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *Proceedings of 8th IEEE Real-Time Systems Symposium*, pp. 166-171. IEEE Computer Society Press, December 1989.

[8] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of ACM*, vol. 20, no. 1, pp. 45-61, January 1973.

[9] J.W.S. Liu, W.K. Shih, K.J. Lin, R. Bettati, and J.Y.Chang, "Imprecise Computations," *Proceedings of IEEE*, vol. 82, no. 1, pp. 83-94, January 1994.

[10] G. Manimaran and C. Siva Ram Murthy, "An Efficient Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 3, pp. 312-319, March. 1998.

[11] G. Manimaran, A. Manikutty, C. Siva Ram Murthy, "DHARMA: A Tool for Evaluating Dynamic Scheduling Algorithms for Real-Time Multiprocessor Systems" *The Journal of Systems and Software,* vol. 50, pp. 131-149, 2000.

[12] K. Ramamritham, J.A. Stankovik, and P.-F. Shian, "Efficient Scheduling Algorithm for Real-Time Multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 184-194, Apr. 1990.

[13] K. Ramamritham and J.A. Stankovic, "Scheduling Algorithms and Operating Systems Support for Real-Time Systems," *Proceedings of IEEE*, vol. 82, no. 1, pp. 55-67, January 1994.

[14] P. Ramamritham, "Graceful Degradation in Real-Time Control Applications Using (m, k)-Firm Guarantee" *Proceedings of FTCS,* pp. 132-141, 1997.

[15] I. Ripoll, A. Crespo and A. García-Fornes, "An Optimal Algorithm for Scheduling Soft Aperiodic Tasks in Dynamic-Priority Preemptive Systems," *IEEE Transactions on software engineering*, vol. 23, no. 6, June 1996.

[16] S. Sáez, J. Vila and A. Crespo, "Soft Aperiodic Task Scheduling on Hard Real-Time Multiprocessor Systems," *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications,* 1998.

[17] K.G. Shin and P. Ramanathan, "Real-time computing: A New Discipline of Computer Science and Engineering," *Proceedings of IEEE*, vol. 82, no. 1, pp. 6-24, January 1994.

[18] M. Spuri and G.C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling," *Proceedings of IEEE Real-Time Systems Symposium* , San Juan, Puerto Rico, December 1994.

[19] M. Spuri and G.C. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *Journal of Real-Time Systems*, vol. 10, no. 2, 1996.

[20] J.A. Stankovic and K. Ramamritham, "The Spring Kernel: A New Paradigm for Real-Time Operating Systems," *ACM Operating Systems Review*, vol. 23, pp. 54-71, July 1989.

[21] J. Stankovic, "Real-Time Computing," *BYTE*, invited paper, pp. 155-160, Aug. 1992.

[22] J. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems," *IEEE Computer*, vol. 28, no. 6, pp. 16-25, June 1995.